LAB#11

STRINGS AND STRING MANIPULATION IN C++

One of the most useful data types supplied in the C++ libraries is the string. A string is a Variable that stores a sequence of letters or other characters, such as **"Hello"** or **"May 10th is my birthday!".** Just like the other data types, to create a **string** we first declare it, then we can store a value in it.

string testString;

testString = "This is a string.";

We can combine these two statements into one line:

string testString = "This is a string.";

Often, we use strings as output, and cout works exactly like one would expect:

cout << testString << endl;

Will print the same result as

cout << "This is a string." << endl;

In order to use the string data type, the C++ string header <string> must be included at the top of the program. Also, you'll need to include using namespace std; to make the short name string visible instead of requiring the cumbersome std::string. (As a side note, std is a C++ *namespace* for many pieces of functionality that are provided in standard C++ libraries. For the purposes of this class, you won't need to otherwise know about namespaces.) Thus, you would have the following #include's in your program in order to use the string type.

```
#include <string>
using namespace std;
```

Basic Operations

Let's go into specifics about the string manipulations you'll be doing the most.

Counting the number of characters in a string. The length method returns the number of characters in a string, including spaces and punctuation. Like many of the string operations, length is a *member function*, and we invoke member functions using *dot notation*. The string that is the receiver is to the left of the dot, the member function we are invoking is to the right, (e.g. str.length()). In such an expression, we are requesting the length from the variable str.

Example Program:

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string small, large;
small = "I am short";
large = "I, friend, am a long and elaborate string
indeed"; cout << "The short string is " << small.length()
<< " characters." << endl;
cout << The long string is " << large.length()
<< " characters." <<
endl; return 0;
}
```

Output:

```
The short string is 10 characters.
The long string is 48 characters.
```

Accessing Individual Characters

Using square brackets, you can access individual characters within a string as if it's a **char** array. Positions within a string **str** are numbered from 0 through **str.length() - 1**. You can read and write to characters within a string using **[ ]**.

Comparing Two Strings

You can compare two strings for equality using the **==** and **!=** operators. Suppose you ask the user for his or her name. If the user is Julie, the program prints a warm welcome. If the user is not Neal, the program prints the normal message. Finally… if the user is Neal, it prints a less enthusiastic response.

example program:

```
#include <string>
#include <iostream>
using namespace std;

int main() {
string myName = "ALI";
while (true) {
cout << "Enter your name (or 'quit' to exit): ";
string userName = getLine();
if (userName == "Ahmed") {
cout << "Hi, Ahmed! Welcome back!" << endl;
} else if (userName == "quit") {
cout << endl;
break;
} else if (userName != myName) {
cout << "Hello, " << userName << endl;
} else {
cout << "Oh, it’s you, " << myName << endl;
}
}
return 0;
}
```

Output

```
Enter your name (or 'quit' to exit): ALI
Oh, it's you, ALI
Enter your name (or 'quit' to exit): Ahmed
Hi, Ahmed! Welcome back!
Enter your name (or 'quit' to exit): quit
```

Appending To a String

C++ strings are wondrous things. Suppose you have two strings, **s1** and **s2** and you want to create a new string of their concatenation. Conveniently, you can just write **s1 + s2,** and you’ll get the result you’d expect. Similarly, if you want to append to the end of string, you can use the **+=** operator. You can append either another string or a single character to the end of a string.

Example Program:

```
#include <string>
#include <iostream>
int main() {
string firstname = "Leland";
string lastname = " Stanford";
string fullname = firstname + lastname; // concat the two
strings fullname += ", Jr"; // append another string fullname
+= '.'; // append a single char
cout << firstname << lastname << endl;
cout << fullname << endl;
return 0;
}
```

Output:

```
Leland Stanford
Leland Stanford, Jr.
```

Searching Within A String

The string member function **find** is used to search within a string for a particular string or character. A sample usage such as **str.find(key)** searches the receiver string **str** for the **key**. The parameter **key** can either be a string or a character. (we say the **find** member function is overloaded to allow more than one usage). The return value is either the starting position where the key was found or the constant **string::npos** which indicates the key was not found. Occasionally, you’ll want to control what part of the string is searched, such as to find a second occurrence past the first. There is an optional second integer argument to **find** which allows you to specify the starting position; when this argument is not given, 0 is assumed. Thus, **str.find(key, n)** starts at position **n** within **str** and will attempt to find **key** from that point on. The following code should make this slightly clearer:

Example Program:

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string sentence = "Yes, we went to Gates after we left the room.";
int firstWe = sentence.find("we"); // finds the first "we"
int secondWe = sentence.find("we", firstWe + 1); // finds "we" in
"went" int thirdWe = sentence.find("we", secondWe + 1); // finds the
last "we" int gPos = sentence.find('G');
int zPos = sentence.find('Z'); // returns string::npos
cout << "First we: " << firstWe << endl; cout <<
"Second we: " << secondWe << endl;
cout << "Third we: " << thirdWe << endl;
cout << "Is G there? ";
cout << (gPos != string::npos ? "Yes!" : "No!") <<
endl; cout << "Is Z there? ";
cout << (wPos != string::npos ? "Yes!" : "No!") << endl;
return 0;
}
```

OUTPUT:

```
First we: 5
Second we: 8
Third we: 28
Is G there? Yes!
Is Z there? No!
```

Extracting Substrings

Sometimes you would like to create new strings by extracting portions of a larger one. The **substr** member function creates substrings from pieces of the receiver string. You specify the starting position and the number of characters. For example, **str.substr(start, length)** returns a new string consisting of the characters from **str** starting at the position **start** and continuing for **length** characters. Invoking this member function does not change the receiver string, as it makes a *new* string with a copy of the characters specified.

Example Program:

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string oldSentence;
oldSentence = "The quick brown fox jumped WAY over the lazy
dog"; int len = oldSentence.length();
cout << "Original sentence: " << oldSentence <<
endl; int found = oldSentence.find("WAY ");
string newSentence = oldSentence.substr(0, found);
cout << "Modified sentence: " << newSentence <<
endl; newSentence += oldSentence.substr(found + 4);
cout << "Completed sentence: " << newSentence << endl;
return 0;
}
```

Output

```
Original sentence: The quick brown fox jumped WAY over the lazy dog
Modified sentence: The quick brown fox jumped
Completed sentence: The quick brown fox jumped over the lazy dog
```

There are a couple of special cases for **substr(start, length)**. If **start** is negative, it will cause a run-time error. If **start** is past the end of the string, it will return an empty string (e.g., **""**). If **length** is longer than the number of characters from the start position to the end of the string, it truncates to the end of the string. If **length** is negative, then the behavior is undefined, so make sure that **length** is always non-negative. If you leave off the second argument, the number of characters from the starting position to the end of the receiver string is assumed.

Modifying a String by Inserting and Replacing

Finally, let's cover two other useful member functions that modify the receiver string. The first, **str1.insert(start, str2),** inserts **str2** at position **start** within **str1**, shifting the remaining characters of **str1** over. The second, **str1.replace(start, length, str2),** removes from **str1** a total of **length** characters starting at the position **start**, replacing them with a copy of **str2.** It is important to note that these member functions do modify the receiver string.

Example Program:

```
#include <string>
#include <iostream>
using namespace std;
int main() {
string sentence = "CS106B is good.";
cout << sentence << endl;
// Insert "kind of" at position 8 in
sentence sentence.insert(7, "is best ");
cout << sentence << endl;
// Replace the 10 characters " is best "
sentence.replace(7, 10, " is best ");
cout << sentence << endl;
return 0;
}
```

Lab Tasks

1. Write a program to find a substring within a string. If found display its starting position.

2. Write program takes a string object from the user and calculates the number of vowels, digits and white-spaces.

3. Write a function to check whether a string "s" is a palindrome or not.

4. Write a program to convert a string in lowercase.

5. Write a C++ program to reverse a given string.

6. Write a C++ program to capitalize the first letter of each word of a given string. Words must be separated by only one space.

7. Write a C++ program to count all the words in a given string.